USAISEC
US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5300

1

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES

AIRMICS

Integrated Office Information System (IOIS) Summary Report:

An Object-Oriented SEM Design/Maintenance Methodology for an Integrated Knowledge Base/Database

ASQB-GM-90-024

MAY 1990

AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188
Exp. Date: Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS
NONE

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

3. DISTRIBUTION/AVAILABILITY OF REPORT
N/A

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
ASQB-GM-90-024

5. MONITORING ORGANIZATION REPORT NUMBER(S)
N/A

6a. NAME OF PERFORMING ORGANIZATION
AIRMICS

6b. OFFICE SYMBOL *(If applicable)*
ASQB-GM

7a. NAME OF MONITORING ORGANIZATION
N/A

6c. ADDRESS (City, State, and Zip Code)
115 O'Keefe Bldg.
Georgia Institute of Technology
Atlanta, Ga 30332-0800

7b. ADDRESS *(City, State, and ZIP Code)*
N/A

8b. NAME OF FUNDING/SPONSORING ORGANIZATION
AIRMICS

8b. OFFICE SYMBOL *(If applicable)*
ASQB-GM

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State, and ZIP Code)*
115 O'Keefe Bldg.
Georgia Institute of Technology
Atlanta, GA 30332-0800

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
62783A	DY10	05	

11. TITLE *(Include Security Classification)*
Integrated Office Information System (IOIS) Summary Report:
An Object-Oriented SEM Design/Maintenance Methodology for an Integrated Knowledge Base/Database

12. PERSONAL AUTHOR(S)
Dr. Olivia R. Liu Sheng, Joline Morrison, Mike Morrison

13a. TYPE OF REPORT

13b. TIME COVERED
FROM_______ TO_______

14. DATE OF REPORT *(Year, Month, Day)*
May 1990

15. PAGE COUNT
28

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUBGROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

An object-oriented Structured Entity Method (SEM) design and maintenance methodology was completed as partial fulfillment of the February 3, 1989 proposal. Due to the object-oriented nature, the methodology has been renamed the Structured Object Methodology (SOM). SOM provides high level design tools somewhat similar to Entity Relationship Diagrams that, unlike ER Diagrams, are top down and hierarchical. SOM diagrams translate easily into objects that can be coded into a knowledge base. The resulting knowledge base is easier to understand and maintain than conventional knowledge bases with no underlying organization to their rules.

As was also suggested in the February 3, 1989 proposal, a prototype of an integrated KB/DB for the IOIS was completed. The prototype fully utilizes SOM and verifies that it is a reasonable design technique. SOM proved to be better suited to the task of integrating a KB/DB than other design techniques that were explored at the same time. A subset of AIRMICS data was modeled using SOM and then entered into a normalized database. Oracle was chosen for the DBMS and the knowledge base was written using PC-Scheme/Scoops, an object-oriented dialect of LISP.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Michael Evans

22b. TELEPHONE *(Include Area Code)*
404/894-3107

22c. OFFICE SYMBOL
ASQB-GM

83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

This research was performed for the Army Institute for Research in Management Information, Communications and Computer Science (AIRMICS), the RDTE organization of the U.S. Army Information Systems Engineering Command (USAISEC). This research is not to be construed as an official Army position, unless so designated by other authorized documents. Material included herein is approved for public release, distribution unlimited. Not protected by copyright laws.

THIS REPORT HAS BEEN REVIEWED AND IS APPROVED

s/ James Gantt
James Gantt
Chief, MISD

s/ John R Mitchell
John R. Mitchell
Director
AIRMICS

IOIS Summary Report:
An Object-Oriented SEM Design/Maintenance Methodology for an Integrated Knowledge Base/Database

Joline Morrison
Mike Morrison
Olivia R. Liu Sheng
University of Arizoria
Tucson, AZ 85721

This research was supported in part by a grant from the Army Institute of Research in Management Information, Communications, and Computer Science (AIRM!CS), Atlanta, Georgia, Grant #: DAKF-11-88-C-0021.

EXECUTIVE SUMMARY

An object-oriented Structured Entity Method (SEM) design and maintenance methodology was completed as partial fulfillment of the February 3, 1989 proposal. Due to its object-oriented nature, the methodology has been renamed the Structured Object Methodology (SOM). SOM provides high level design tools somewhat similar to Entity Relationship Diagrams that, unlike ER Diagrams, are top down and hierarchical. SOM diagrams translate easily into objects that can be coded into a knowledge base. The resulting knowledge base is easier to understand and maintain than conventional knowledge bases with no underlying organization to their rules.

As was also suggested in the February 3, 1989 proposal, a prototype of an integrated KB/DB for the IOIS was completed. The prototype fully utilizes SOM and verifies that it is a reasonable design technique. SOM proved to be better suited to the task of integrating a KB/DB than other design techniques that were explored at the same time. A subset of AIRMICS data was modeled using SOM and then entered into a normalized database. Oracle was chosen for the DBMS and the knowledge base was written using PC-Scheme/Scoops, an object-oriented dialect of LISP.

1. INTRODUCTION

Data may be characterized as factual, atomic statements, that occur in large volumes. Conversely, knowledge consists of imprecise statements applicable to large groups of facts[1]. Data is generally more dynamic than knowledge; for example, a person's address (data) may change, but the procedure of finding his house by looking up the address (knowledge) on a city map does not.

A database uses a database management system (DBMS) to efficiently manipulate a repository of data. Likewise, a knowledge base (KB) uses an inference engine to efficiently manipulate knowledge (usually in the form of rules) and data (often hard-coded as facts within rules). The coupling of DBMS and KB technologies allows facts to be retrieved from a database and acted upon by a KB. Complex data retrieval tasks involving abstract knowledge are assisted by the KB. Most coupled KB/DBMS systems have not overcome one of the fundamental differences between data and knowledge, however: the dynamic nature of data. If database tables or schemas are revised, knowledge base rules must be updated to reflect these changes. Databases can be easily updated with new tables, fields and records; however, reflecting such additions in a knowledge base is difficult.

The purpose of this paper is to demonstrate a solution to this problem with a prototype of an intelligent database that explores object-oriented database and knowledge base design within a dynamic query environment. This prototype, called PATHFINDER, allows users with little system or database knowledge to access all possible combinations of database information easily and quickly. PATHFINDER's KB is easily modified to reflect changes in the database schema. This paper provides a brief review of intelligent database theory and previous intelligent database systems, followed by a description of PATHFINDER's architecture, design and implementation. Ideas for additional features and functions are also presented.

2. INTELLIGENT DATABASE THEORY

An intelligent database is separated into two parts: the "extensional" database and the "intensional" database[2]. The extensional database stores explicit data values while the intensional database stores deductive information used to derive new data from the explicit data. An example of tuples in an extensional BUILDING_SUPPLIER table could be represented in the predicate form of (TABLE_NAME(FIELD1, FIELD2, ...)) by:

BUILDING_SUPPLIER(ACME,PAINT)
BUILDING_SUPPLIER(SMITH,LUMBER)
BUILDING_SUPPLIER(ACE, PAINT)

A first order logic derived query could be formed to find all suppliers of PAINT or LUMBER. An example of intensional knowledge stating that every SUPPLY carried by a BUILDING_SUPPLIER is a BUILDING_SUPPLY would be:

BUILDING_SUPPLY(y) <--- BUILDING_SUPPLIER(x, y)

The difference between intensionai and extensional knowledge is not always clear, because databases sometimes incorporate features that are intensional by the above definition. Nevertheless, a knowledge base should contain knowledge that is more stable and at a higher level of abstraction that the extensional knowledge contained in a database.

Wiederhold used levels of abstraction to define nine categories of knowledge relevant to a database [3]. They were loosely grouped into intensional and extensional knowledge. A natural place to begin a knowledge base/database coupling is with knowledge one level of abstraction higher than normally found in database. Using Wiederhold's classification, *structural knowledge* is the first level of knowledge that is clearly in the domain of a knowledge base. It is composed primarily of knowledge about dependencies and constraints among the data.

Gardarin and Valduriez[4] base/database integration: *loose coupling, tight coupling*, and *complete integration*. *Loose coupling* is demonstrated by using a call interface to allow communication between an existing relational DBMS and a separate existing logic programming language. The DBMS is invoked using predicates pre-defined in a special syntax, usually SQL. Al-Zobaidie and Grimson[5] also stipulate that all data is loaded to the expert system as a snapshot from the database prior to operation of the expert system (KB). *Tight coupling* relaxes the pre-defined predicate constraint for DBMS calls and makes the DBMS invisible to the user, but the logic programming language and DBMS still exist as separate entities. Al-Zobaidie and Grimson perhaps state this more clearly by saying that data is only retrieved from the database as needed during operation of the expert system.

Complete integration, the highest level of synthesis, requires the logic programming language to be incorporated into the DBMS. Bell, et. al.[2] subdivided tight coupling depending upon whether it occurred at the logical, functional, or physical level. Their functional and physical tight coupling appear to correspond respectively to Gardarin and Valduriez's tight coupling and complete integration.

Al-Zobaidie and Grimson[5] have another way of categorizing types of deductive databases. They divide them into *intelligent DB*, *enhanced ES* (expert system), and *inter-system communication*. Their major distinction lies in whether the underlying design emphasizes a database (with expert system capabilities added), an expert system (with database functionality added), or both. Inter-system communication allows both the KB and DB to continue to operate independently of each other but to also communicate as needed.

3. EXAMPLES OF INTELLIGENT DATABASES

The KDL-ADVISOR was a system built by Potter[6] to demonstrate the feasibility of the Knowledge/Data Model. A schema specification language is used to capture both knowledge semantics as well as data semantics. A number of extensions to the semantic data model are presented all of which are intended to allow the representation of more abstract knowledge. The KDL-ADVISOR knowledge base was coded using Prolog. The DBMS was not mentioned and possibly extensional facts were also coded using Prolog. As a result, this prototype may not be comparable to PATHFINDER and possibly does not represent an integrated KB/DB. Potter presents a simple example to illustrate the ADVISOR's knowledge/data schema. The example shown appeared quite complex, however, when diagrammed and was difficult to follow. Perhaps the Knowledge/Data Model would be well suited to professionals specializing in this methodology, but for others, a simpler model like PATHFINDER's would help.

LOOD (Logic-oriented object data base) was described by Sheu[7] and is a way to formally describe intelligent databases. Sheu notes "a logic-oriented object base can be described as a data model that is constructed on the basis of an enhanced semantic data model that incorporates more procedural semantics and is represented in logic." No mention was made of a prototype using the LOOD methodology, however, it appears that one could be constructed. LOOD appears to be more mathematically oriented than the Knowledge/data Model and as such is even more complex. For this reason, it will probably not become widely used.

Others have described systems without ꞡoing into the underlying formalisms in as great depth. Smith, et. al[8, 9] have described a knowledge based search intermediary called EP-X that is designed to facilitate information retrieval from a thesaurus type of system. Documents were represented as frames with a list of title, authors, title, etc. Concepts (keywords, subjects, etc.) were also organized hierarchically to support efficient searching for relevant documents. The tools used to construct the KB and DB were not described in the two papers reviewed. No information as to the type of coupling was presented.

CoalSORT[10] can be categorized as a browsing knowledge-based interface to a database. It uses a frame based semantic net to aid both searchers and indexers when seeking or cataloging documents. CoalSORT's method of browsing is similar to PATHFINDER's. Its KB was written with FrameKit, a frame representation language written at Carnegie Mellon University. A DBMS was not mentioned.

4. DESIGN AND IMPLEMENTATION

PATHFINDER is part of the University of Arizona MIS Department's Integrated Office Information System (IOIS) project. Although most office systems are used primarily by clerical and secretarial employees, the IOIS aims to increase office productivity by creating a system for use by both managerial and clerical/secretarial personnel. A wide range of applications, from word processing to distributed electronic meeting systems, are integrated into a system with a consistent user interface. PATHFINDER is intended to provide database support for all IOIS applications.

PATHFINDER was developed on an 80386 PC, and eventually will be ported to heterogenous hardware and operating systems. Oracle was chosen as the DBMS because of its ability to make dynamic queries. C was chosen as the interface language to achieve compatibility with Oracle's Pro C pre-compiler, which allows the dynamic queries through SQL calls embedded in C programs. PC Scheme with SCOOPS (*Sc*heme *O*bject-*O*riented *P*rogramming *S*ystem), a LISP dialect, was chosen for the knowledge base implementation because of its object-oriented features and because of the authors familiarity with it.

The current prototype allows users to navigate through a relational database, determine related entities, and make queries involving joins of up to three relations (joins of more than three relations can be done and will be implemented later). The KB requires no modification if fields are added or deleted in the database. If the database schema is changed, the KB is modified using a high level tool written in Scheme that creates new KB objects and stores them in files; when the system is started, these objects are automatically loaded into PATHFINDER's knowledge base.

4.1 System Architecture

PATHFINDER can be characterized as a tightly-coupled system [4, 5] and also fits Al-Zobaidie and Grimson's[5] definition of a type 3c deductive database (inter-system communication with interfacing routines lying between the user, KB, and DBMS). Figure 1 shows a high level view of PATHFINDER's architecture, and illustrates that the interface is the communications link between the knowledge base, database, and system users and administrators. Users are required to go through the knowledge base to access the database; the database and knowledge base administrators can access the database or knowledge base directly through the interface to make system modifications. The primary use of databasc facts returned to the knowledge base has been to supply display and search field names; database query results are returned directly to the user via the interface.

Oracle's pre-compiler, Pro C, enables Oracle calls to be embedded within C programs. Pro C also provides for "Dynamic SQL", which allows dynamically defined SQL statements (i.e. statements unknown at compile time) to be executed. Four levels of Dynamic SQL are available, distinguished by incrementally-increased query flexibility. The simplest Dynamic SQL program requires most of a SQL call to be anticipated in advance and coded into the program; the most complex, used by PATHFINDER, allows any SQL call to specified entirely at runtime.

4.2 Interface

4.2.1 Functions PATHFINDER's first menu displays high-level database object names representing specific query topics, and prompts the user to select names of topics involved in a query (Figure 2-1). The user is allowed to choose several names from each menu, and may sometimes choose a group of unrelated names (topics). If this happens, an error message is displayed and the system backs up to the previous state. If a query can be made using the selected name or combination of names, the user chooses SELECT SEARCH/DISPLAY FIELDS and proceeds with the query (Figure 2-2), or when available, SELECT MORE CRITERIA, to add more names.

If the user chooses SELECT SEARCH/DISPLAY FIELDS, the system dynamically queries the database to determine the relevant fields, and allows the user to specify display fields and search conditions (Figures 3-1 and 3-2). The text of the resulting query is then

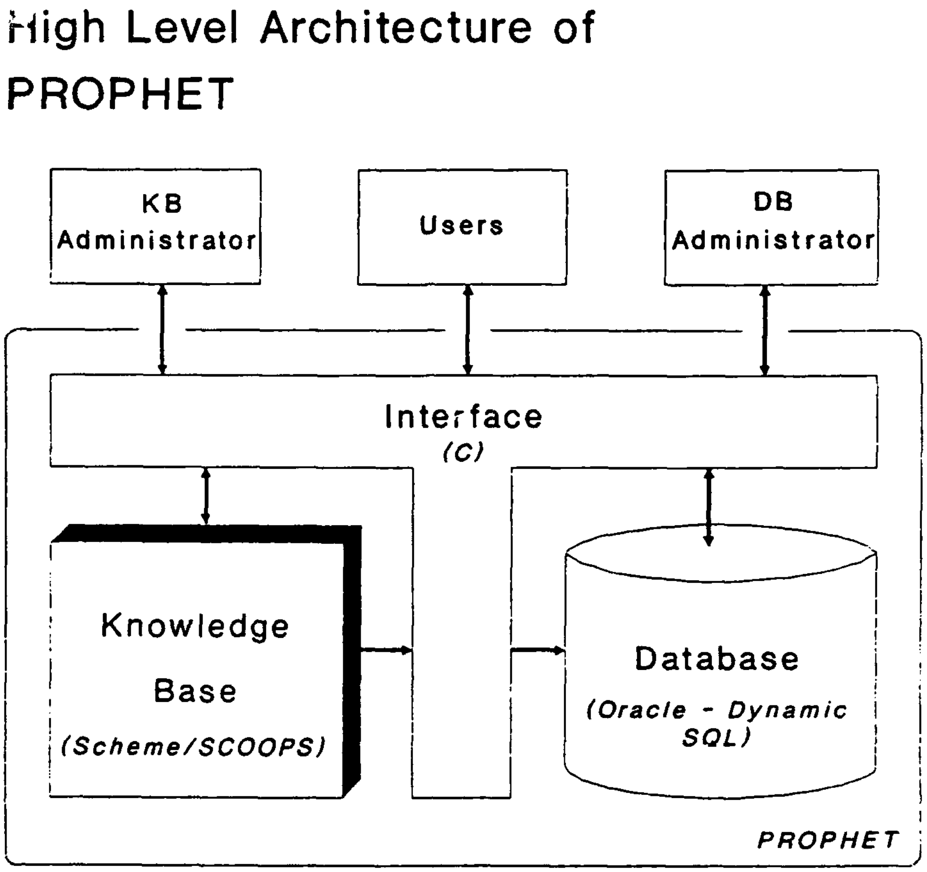


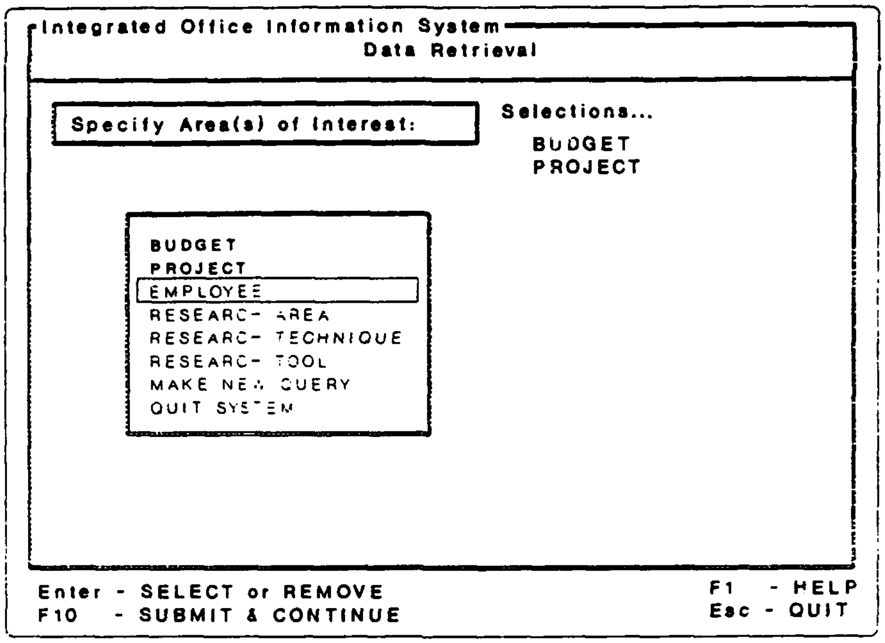
Figure 1

displayed (Figure 4-1), the query is run, and the result is shown in a scrollable window (Figure 4-2). The initial menu is then re-displayed.

If the user chooses to add more object names, all remaining related objects are displayed. He or she may continue to select related objects as long as any are available, or may specify to select display/search fields and then run a query. The user is always allowed to make a new query or quit the system.

4.2.2 Components The interface consists of two high-level modules, the User/KB Interface and the User/Database Interface (Figure 5). This was necessary because the Oracle DBMS and the window libraries were incompatible; two window libraries were tried.

(1)



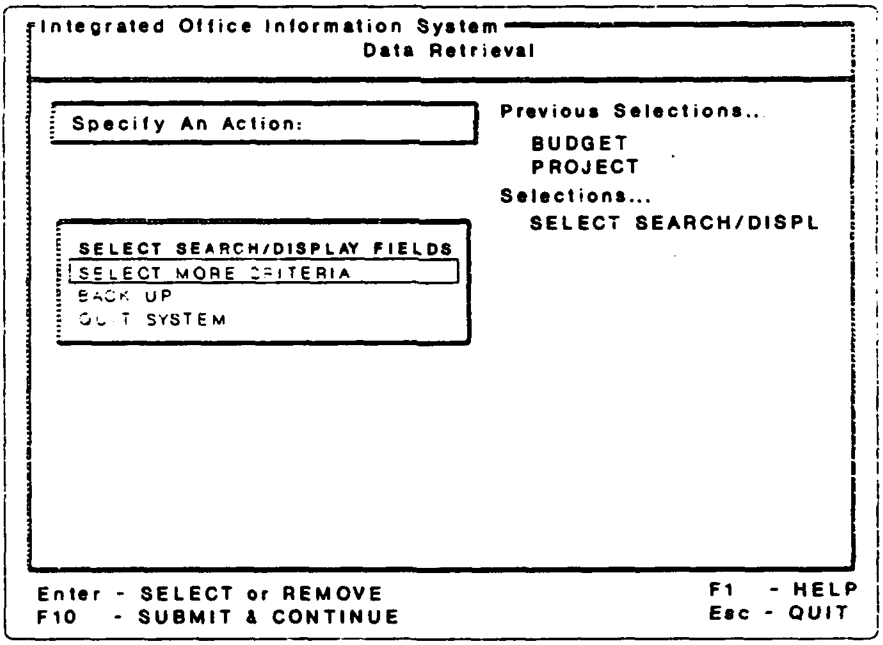
```
Integrated Office Information System
                         Data Retrieval

 Specify Area(s) of Interest:          Selections...
                                         BUDGET
                                         PROJECT

      BUDGET
      PROJECT
      EMPLOYEE
      RESEARCH AREA
      RESEARCH TECHNIQUE
      RESEARCH TOOL
      MAKE NEW QUERY
      QUIT SYSTEM

Enter - SELECT or REMOVE                          F1  - HELP
F10   - SUBMIT & CONTINUE                         Esc - QUIT
```

(2)



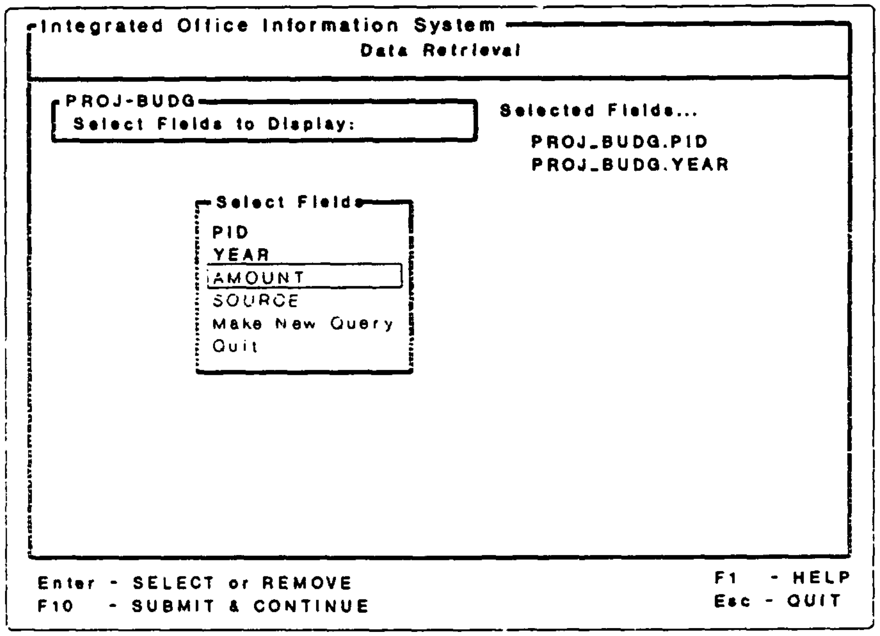
```
Integrated Office Information System
                         Data Retrieval

 Specify An Action:                    Previous Selections..
                                         BUDGET
                                         PROJECT
                                       Selections...
                                         SELECT SEARCH/DISPL
   SELECT SEARCH/DISPLAY FIELDS
   SELECT MORE CRITERIA
   BACK UP
   QUIT SYSTEM

Enter - SELECT or REMOVE                          F1  - HELP
F10   - SUBMIT & CONTINUE                         Esc - QUIT
```

Figure 2

(1)



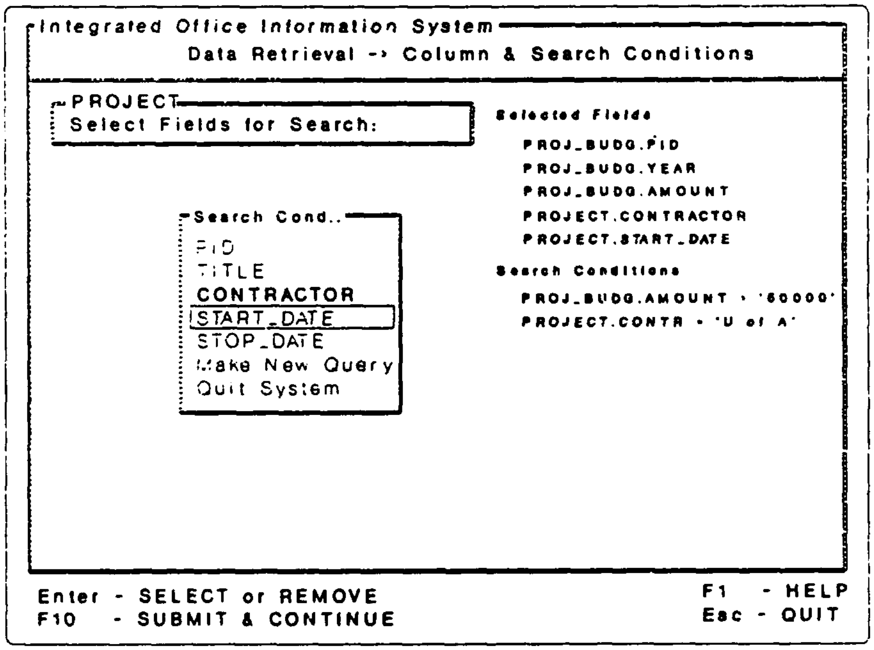
```
Integrated Office Information System
                              Data Retrieval

PROJ-BUDG                                      Selected Fields...
 Select Fields to Display:                        PROJ_BUDG.PID
                                                  PROJ_BUDG.YEAR

              Select Fields
               PID
               YEAR
               AMOUNT
               SOURCE
               Make New Query
               Quit

Enter - SELECT or REMOVE                                  F1  - HELP
F10   - SUBMIT & CONTINUE                                 Esc - QUIT
```

(2)



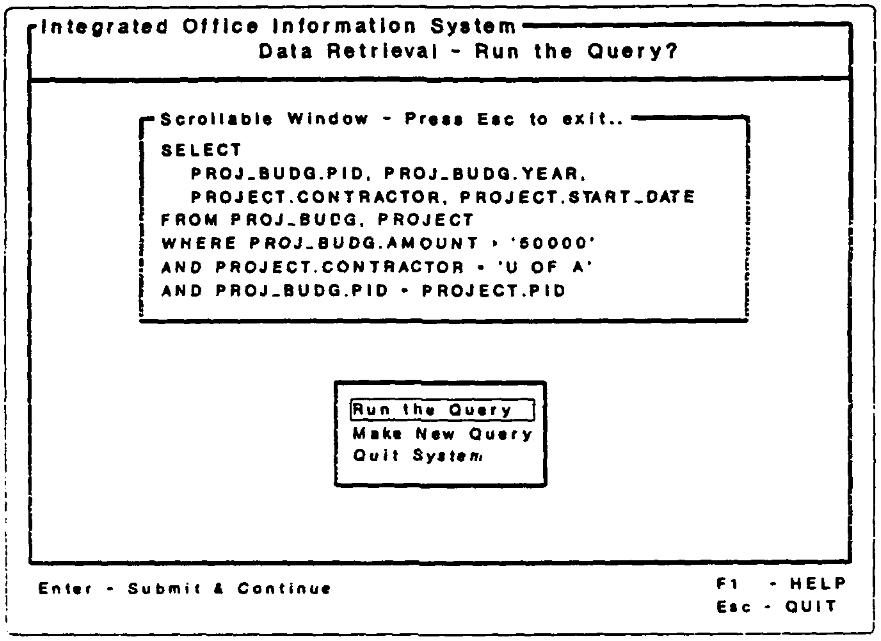
```
Integrated Office Information System
              Data Retrieval -> Column & Search Conditions

PROJECT                                        Selected Fields
 Select Fields for Search:                        PROJ_BUDG.PID
                                                  PROJ_BUDG.YEAR
                                                  PROJ_BUDG.AMOUNT
             Search Cond..                        PROJECT.CONTRACTOR
              PID                                 PROJECT.START_DATE
              TITLE                            Search Conditions
              CONTRACTOR                          PROJ_BUDG.AMOUNT > '60000'
              START_DATE                          PROJECT.CONTR = 'U of A'
              STOP_DATE
              Make New Query
              Quit System

Enter - SELECT or REMOVE                                  F1  - HELP
F10   - SUBMIT & CONTINUE                                 Esc - QUIT
```

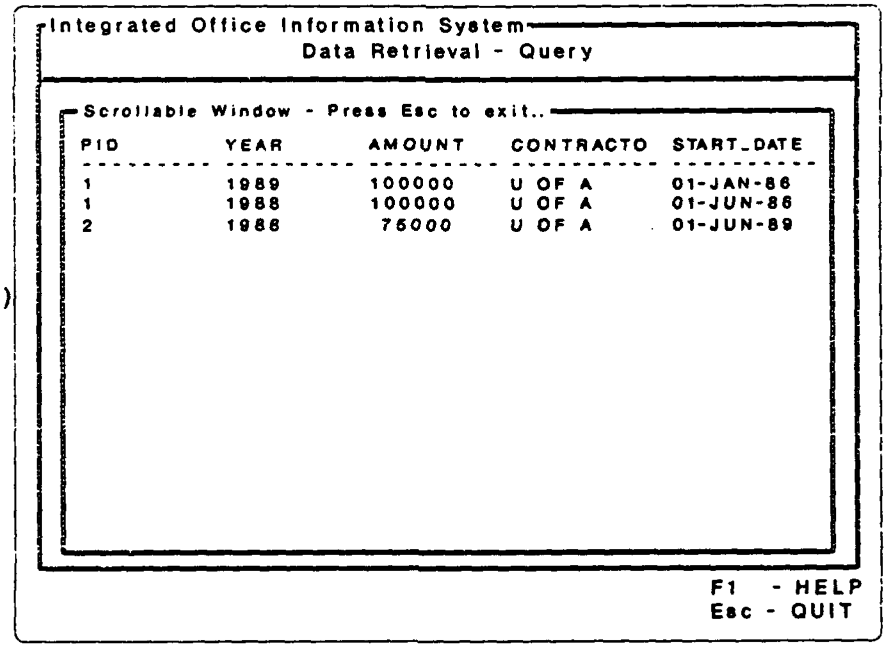
Figure 3

(1)


Integrated Office Information System
Data Retrieval - Run the Query?
Scrollable Window - Press Esc to exit..
Run the Query
Make New Query
Quit System
Enter - Submit & Continue
F1 - HELP
Esc - QUIT

```
SELECT
   PROJ_BUDG.PID, PROJ_BUDG.YEAR,
   PROJECT.CONTRACTOR, PROJECT.START_DATE
FROM PROJ_BUDG, PROJECT
WHERE PROJ_BUDG.AMOUNT › '50000'
AND PROJECT.CONTRACTOR • 'U OF A'
AND PROJ_BUDG.PID • PROJECT.PID
```

(2)


Integrated Office Information System
Data Retrieval - Query
Scrollable Window - Press Esc to exit..
F1 - HELP
Esc - QUIT

```
PID          YEAR         AMOUNT       CONTRACTO    START_DATE
----------   ----------   ----------   ----------   ----------
1            1989         100000       U OF A       01-JAN-86
1            1988         100000       U OF A       01-JUN-86
2            1988          75000       U OF A       01-JUN-89
```

Figure 4

The interface was therefore split into separate types of executable programs - one type reads files created by the knowledge base and then interacts with the user, and the other type accesses the DBMS. C's *spawnl* command (similar to *fork* and *execl*) was used to get from one executable file to another.

Rather than putting all of the User/KB Interface into one executable file, three smaller files were created: *Route, Display/Search*, and *Query*. Smaller files load faster from the knowledge base's Scheme DOS-CALL command (equivalent to a spawnl with a P-WAIT parameter). Two separate executable files were created for the User/Database Interface.

Scheme's object-oriented functions conflicted with Oracle when they were loaded into extended memory. This problem was resolved with only a slight performance penalty by running Scheme in the lower 640 KB of memory.

4.3 Database Design and Implementation

The current IOIS prototype is designed for the office of AIRMICS, the IOIS project's funding agency. PATHFINDER was therefore designed according to a subset of their database specifications. AIRMICS is an information systems research office of the United States Army. One of their tasks is to review research proposals, fund projects that cover topics relevant to their current goals, and oversee these projects. Employees at AIRMICS include researchers, managers, and staff support personnel. When a project is approved for funding, a manager and researchers may be assigned to it. Every project falls into a single Research Area (such as Office Automation); projects may employ several Research Techniques (such as case study, survey, software engineering, etc.) and Research Tools (such as KEE, LISP, Excelerator, etc.). Researchers and managers specialize in certain Research Areas, Techniques, and Tools. Further, Research Areas correspond to certain Research Techniques and Research Tools, and vice-versa. Every project has a specific budget; AIRMICS' total budget consists of their Operations Budget plus the total of all of the project budgets.

The database schema was designed using both the Structured Object Methodology (SOM)[11] and the ER Model[12]. SOM was developed by Higa[11] as a graphic method of designing database schemas by showing hierarchical relationships. This method requires the creation of two diagrams, the Initial SOM and Final SOM. The Initial SOM presents the logical relations between entities (Figure 6); the Final SOM represents the logical file structure (Figure 7). Rules similar to those used to generate files from an ER Model can be used generate a Final SOM from an Initial SOM. In general, one-to-one relationships

Interface Components

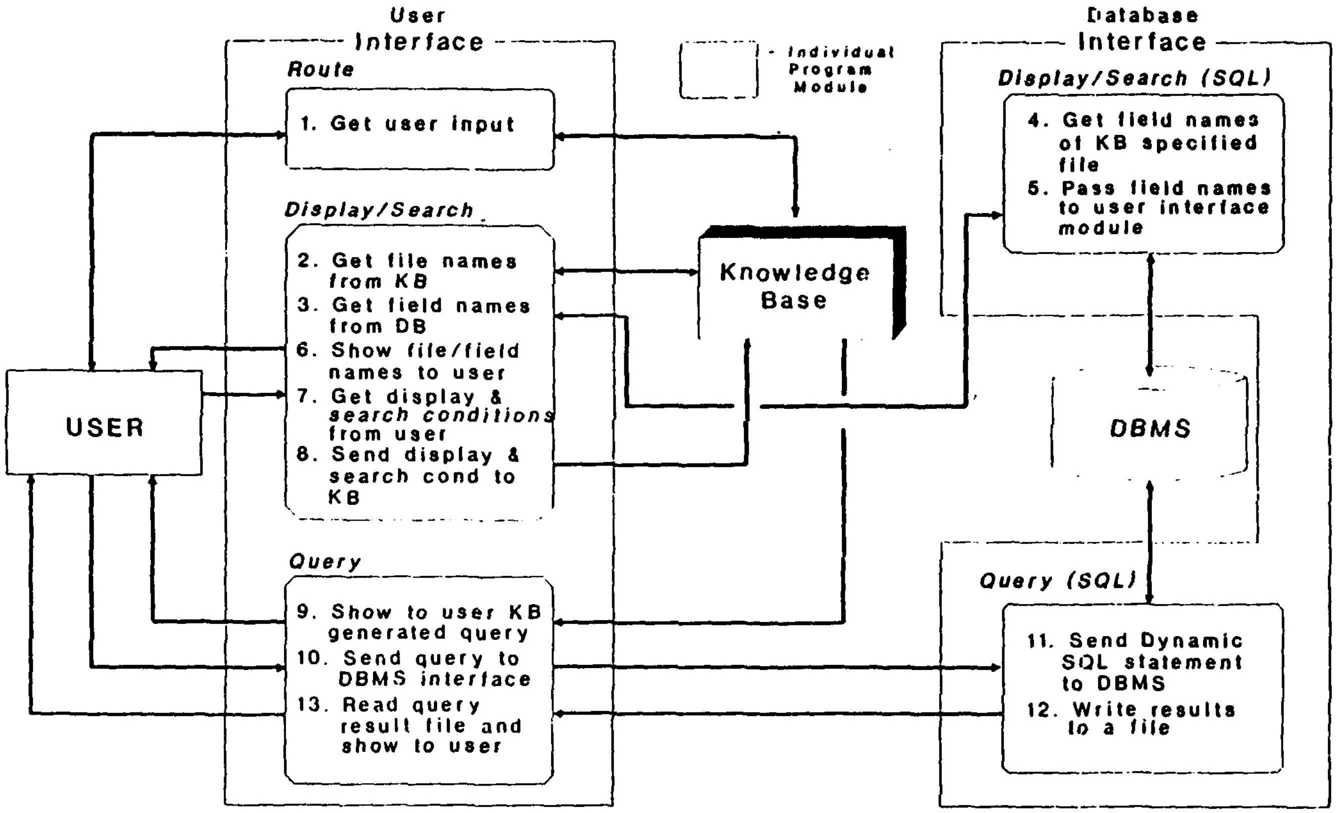

User Interface
Route
1. Get user input
Display/Search
2. Get file names from KB
3. Get field names from DB
6. Show file/field names to user
7. Get display & search conditions from user
8. Send display & search cond to KB
Query
9. Show to user KB generated query
10. Send query to DBMS interface
13. Read query result file and show to user
USER
- Individual Program Module
Knowledge Base
Database Interface
Display/Search (SQL)
4. Get field names of KB specified file
5. Pass field names to user interface module
DBMS
Query (SQL)
11. Send Dynamic SQL statement to DBMS
12. Write results to a file

Figure 5

generate one file, one-to-many relationships generate two files, and many-to-many relationships generate three files [13]. The resulting database schema is in third normal form. (A more detailed description of SOM is in [14]). Although we were initially more familiar with the ER Model, SOM was easier to translate into KB objects and was easier to follow as relationships became more complex. Figures 8 and 9 show ER diagrams comparable to the SOM diagrams and are included for comparison.

4.4 Knowledge Base Design and Implementation

The components of the Knowledge Base (KB) are shown in Figure 10, and the functions of each component are explained below.

4.4.1 KB-Objects An instance (object) of KB-Objects exists for each possible combination of database entity names that a user may select to be used in a query. These objects are a unique type of object called a *par-set*[15]. These objects have methods that allow all instances of the objects to be evaluated concurrently. This property is necessary in this application because when the user selects a desired combination of entity names, the *KB-Controller* sends a message to each KB-OBJECT asking if it is the requested object. In a large data base with several entities, hundreds of entity-combination objects can potentially exist. Concurrent evaluation is necessary to make system response time acceptable.

Each of these objects have the following instance variables that uniquely identify them, determine the database structure, and provide query knowledge:

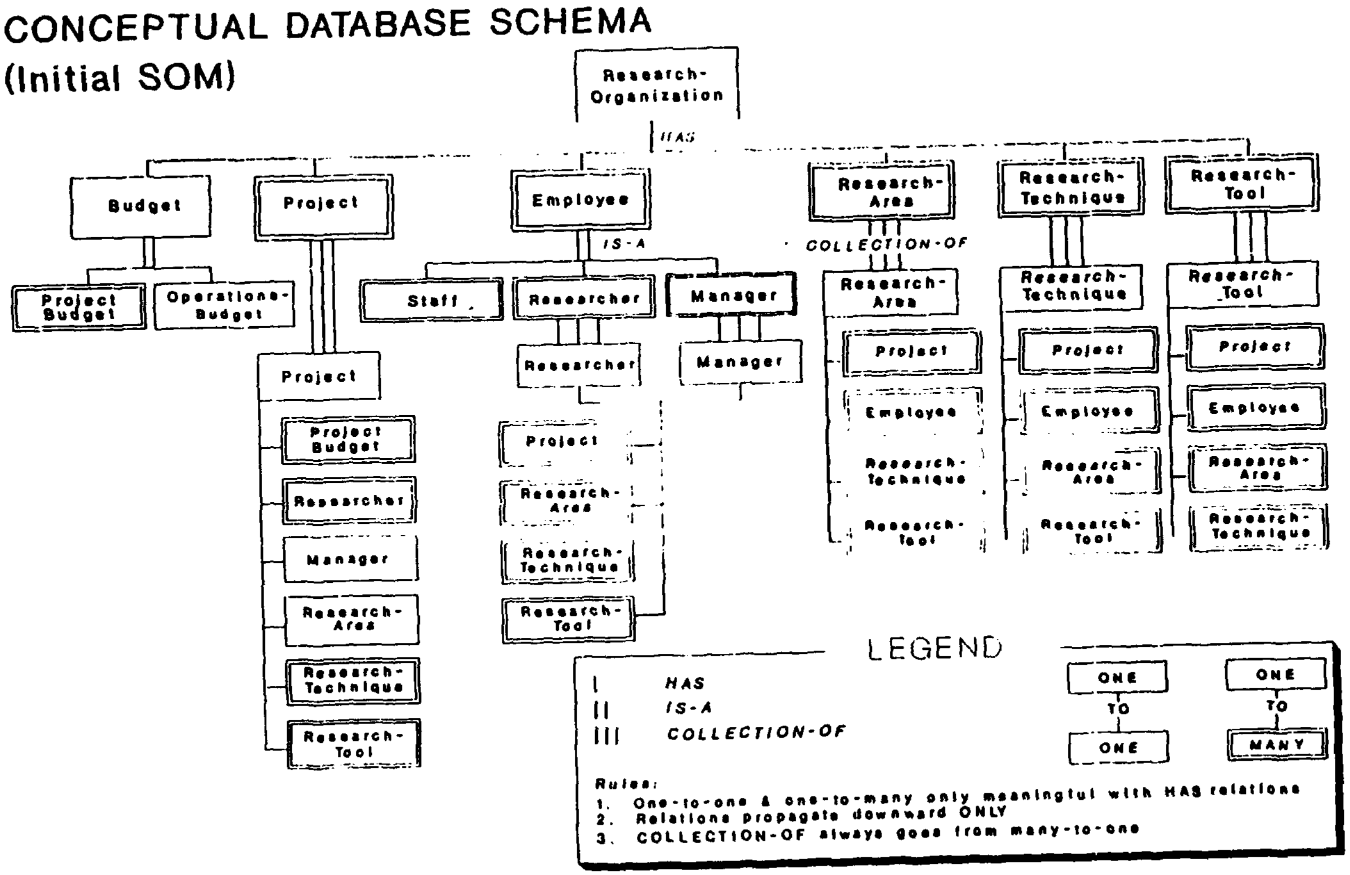
Determinant-List - a list of object names that uniquely identify the object

Child-List - a list of objects directly related to the current object

Data-Files - a list of the database file names related to the object

Keys - a list of the keys of the files specified in the Data-Files

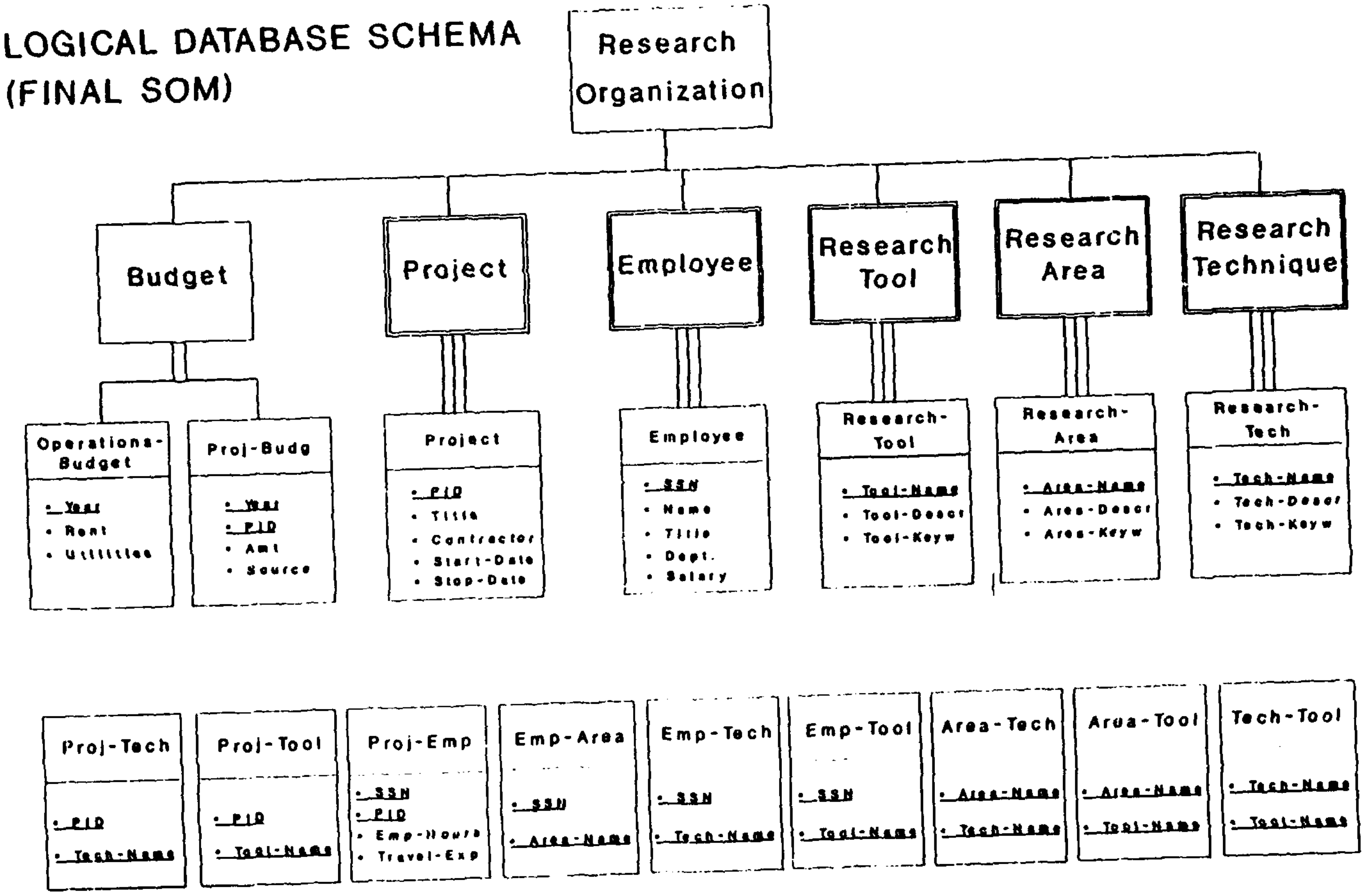
Link-File - a list of the linking files related to the object; for example, in the object PROJECT-EMPLOYEE, data files PROJECT (with key PID) and EMPLOYEE (with key SSN) are linked by file PROJ-EMP (with composite key PID and SSN)



CONCEPTUAL DATABASE SCHEMA
(Initial SOM)

Rules:
1. One-to-one & one-to-many only meaningful with HAS relations
2. Relations propagate downward ONLY
3. COLLECTION-OF always goes from many-to-one

Figure 6



LOGICAL DATABASE SCHEMA
(FINAL SOM)

Figure 7

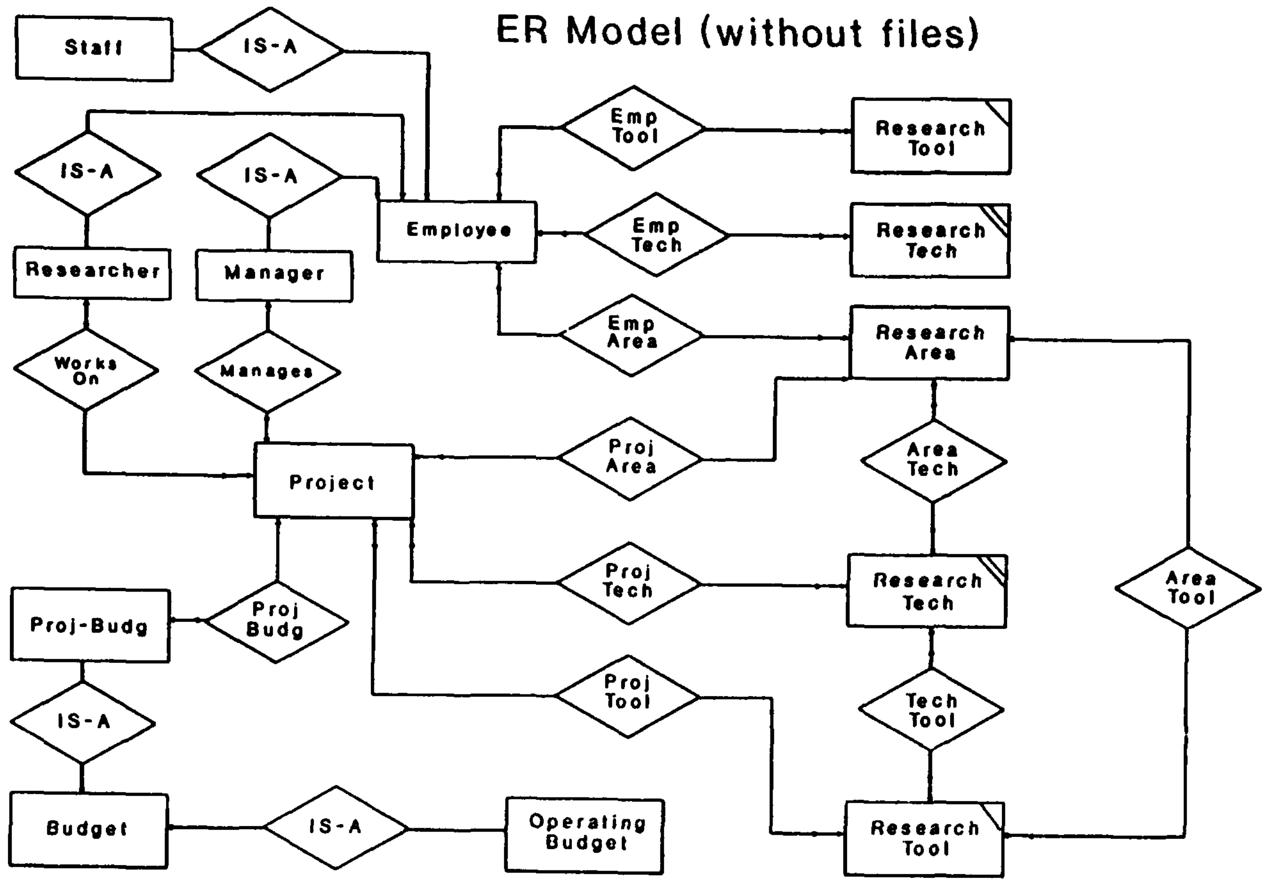

ER Model (without files)
Staff
IS-A
IS-A
IS-A
Researcher
Manager
Employee
Emp Tool
Emp Tech
Emp Area
Research Tool
Research Tech
Research Area
Works On
Manages
Project
Proj Area
Area Tech
Proj Tech
Research Tech
Area Tool
Proj-Budg
Proj Budg
Proj Tool
Tech Tool
IS-A
Budget
IS-A
Operating Budget
Research Tool

Figure 8

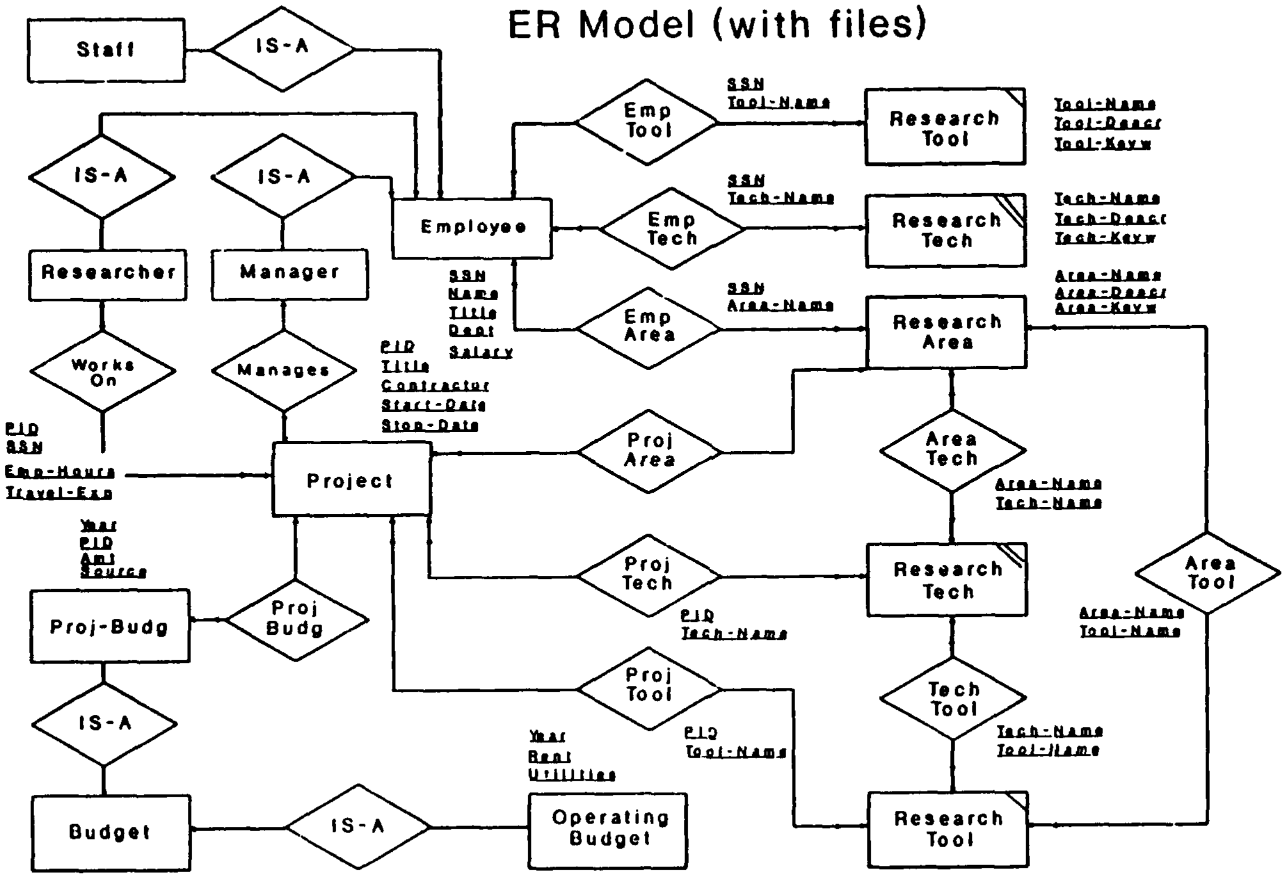

ER Model (with files)
Staff
IS-A
IS-A
IS-A
Researcher
Manager
Employee
Works On
Manages
Project
Emp Tool
Emp Tech
Emp Area
SSN
Tool-Name
SSN
Tech-Name
SSN
Area-Name
Research Tool
Tool-Name
Tool-Descr
Tool-Keyw
Research Tech
Tech-Name
Tech-Descr
Tech-Keyw
Research Area
Area-Name
Area-Descr
Area-Keyw
SSN
Name
Title
Dept
Salary
PID
Title
Contractor
Start-Date
Stop-Date
PID
SSN
Emp-Hours
Travel-Exp
Proj Area
Area Tech
Area-Name
Tech-Name
Area Tool
Area-Name
Tool-Name
Proj Tech
PID
Tech-Name
Research Tech
Tech Tool
Tech-Name
Tool-Name
Proj Tool
PID
Tool-Name
Research Tool
Year
PID
Amt
Source
Proj-Budg
Proj Budg
IS-A
Budget
IS-A
Operating Budget
Year
Rent
Utilities

Figure 9

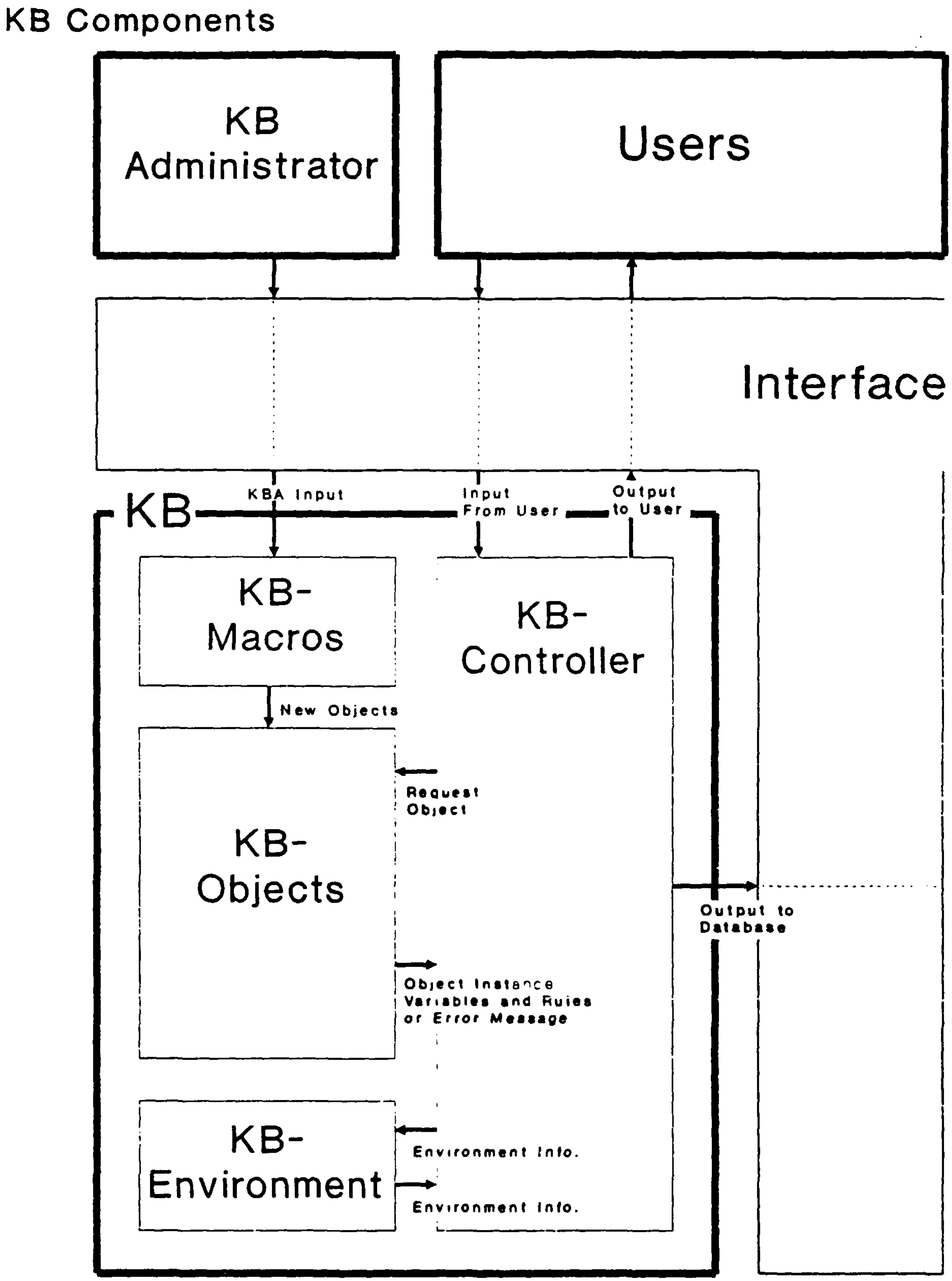

KB Components
KB Administrator
Users
Interface
KB
KBA Input
Input From User
Output to User
KB-Macros
KB-Controller
New Objects
Request Object
KB-Objects
Output to Database
Object Instance Variables and Rules or Error Message
KB-Environment
Environment Info.
Environment Info.

Figure 10

Each KB-Objects also contains another par-set object consisting of a list of rules that determines the object's current position in the inferencing sequence and, when fired, carries out the appropriate action. A summary of rule conditions and actions is as follows:

Condition	Action
Object Initially Selected	Send menu items SELECT SEARCH/DISPLAY FIELDS, get MORE CRITERIA, BACK-UP or QUIT to interface
User chooses SELECT SEARCH/ DISPLAY FIELDS	Send database file names to interface; interpret them, determine type of query, get the data file keys if necessary, and assemble the query
User chooses to get MORE CRITERIA	Send object's Child-List to interface

The hierarchical nature of the SOM translates easily into these objects representing

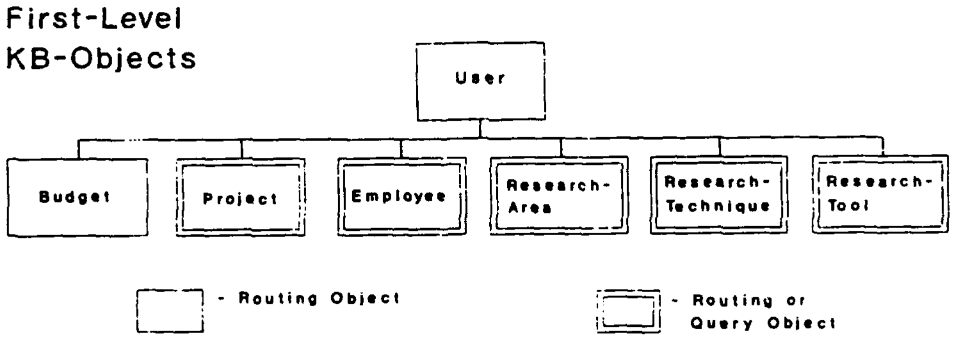


Figure 11

user-defined queries. The first level of KB-Objects (Figure 11) was created to represent the first-level objects on the Initial SOM. For these first-level objects, direct queries can be made only through objects that correspond to a single database file. From the Final SOM

(Figure 7), this includes all first-level objects except BUDGET[1]. At this point, two types of KB-Objects emerge: Route-Only-Objects, and Route-or-Query-Objects.

Next, additional objects were created to represent the expansion of the first level objects to include all possible combinations of related objects that might be used when making a query. The Initial SOM shows database object relationships hierarchically, and is used to determine the next level of objects. For example, the expansion of related object-name-combinations for *BUDGET* includes *BUDGET-PROJECT, BUDGET-OPERATIONS*, and *BUDGET-PROJECT-OPERATIONS*, since *PROJECT* and *OPERATIONS* are *BUDGET'S* only children. Objects resulting from *PROJECT-BUDGET* include combinations of itself with the children of *PROJECT (RESEARCHER, MANAGER, RESEARCH-AREA, RESEARCH-TECHNIQUE*, and *RESEARCH-TOOL)* less *PROJECT-BUDGET* itself, and all possible combinations of these objects. The complete expansion of *BUDGET* is shown in Figures 12 and 13. This expansion reveals a third type of Name-Combination object: the Query-Only-Object. No further expansion is possible beyond this, since no more related database objects exist. Expansion of the remaining first-level objects is completed in a similar manner.

The different types of KB-Objects are differentiated by the values of their instance variables: a Route-Only-Object has no Data-Files, and a Query-Only-Object has an empty Child-List.

A unique instance of KB-Objects, the User-Object, was created to initialize the state variables and the stack that saves environments, and start the inferencing process.

4.4.2 KB-Controller The KB-Controller is the module that coordinates the actions of the other modules in the knowledge base. It interprets user inputs through the Interface through three methods. *Get-Input* assembles the object name combinations chosen by the user, and then asks all KB-Objects if their Determinant-List matches this list. If a match is found, that object's rules are used by the KB-Controller for the next forward-chaining sequence. The object's rule conditions are tested against the present system state. If a condition is satisfied, the selected rule is fired; these rules evoke various interface programs and pause the KB. The interface program prompts the user to input further data, and the cycle is repeated. If no match is found, an error message is given, and the prior

[1] When a user selects *BUDGET*, he or she must further specify *PROJECT, OPERATIONS*, or both, to make a query. However, if *PROJECT* is chosen first, it is possible to either run a query immediately or to be routed to related objects.

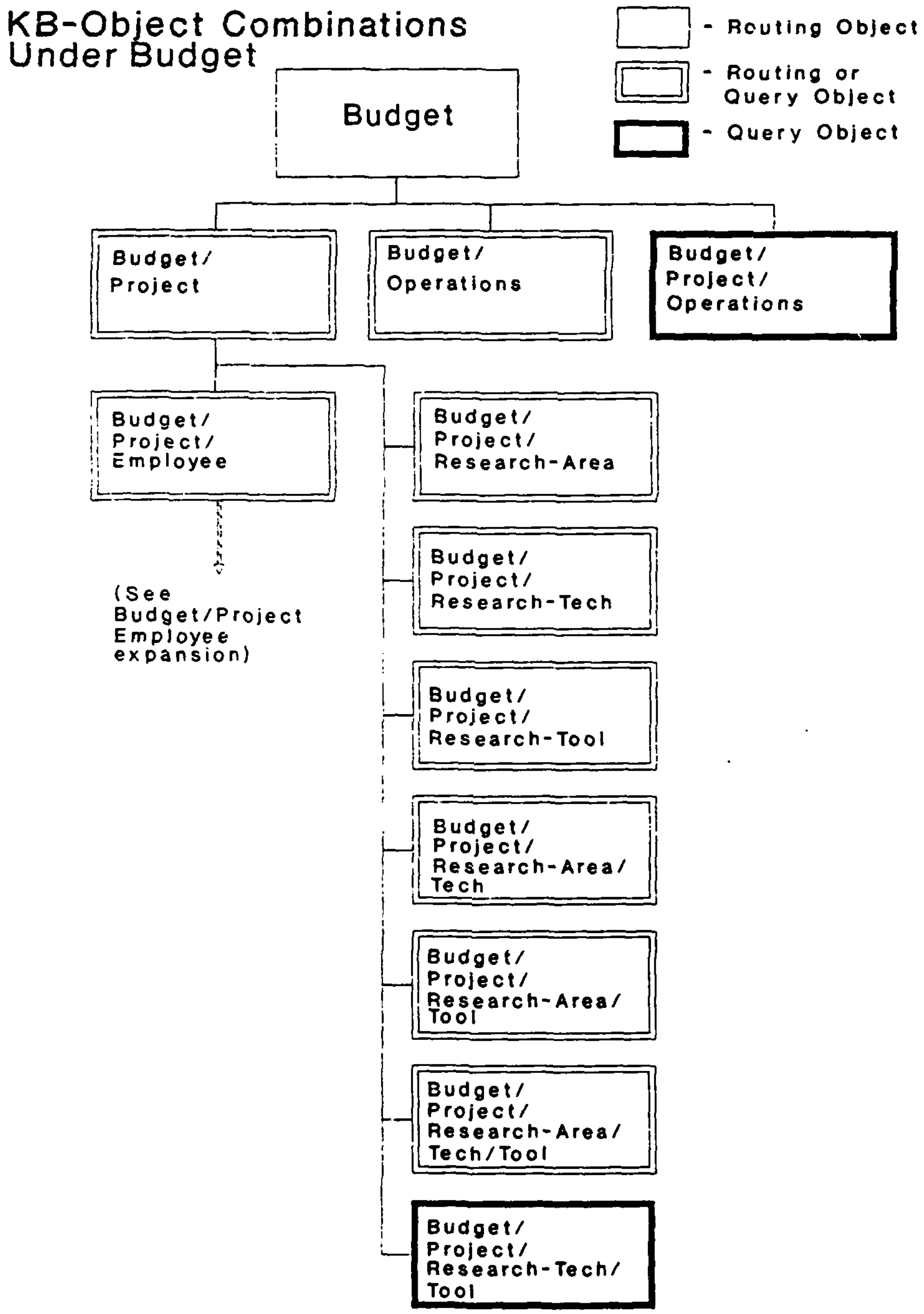

KB-Object Combinations
Under Budget
- Rcuting Object
- Routing or Query Object
- Query Object
Budget
Budget/ Project
Budget/ Operations
Budget/ Project/ Operations
Budget/ Project/ Employee
(See Budget/Project Employee expansion)
Budget/ Project/ Research-Area
Budget/ Project/ Research-Tech
Budget/ Project/ Research-Tool
Budget/ Project/ Research-Area/ Tech
Budget/ Project/ Research-Area/ Tool
Budget/ Project/ Research-Area/ Tech/Tool
Budget/ Project/ Research-Tech/ Tool

Figure 12

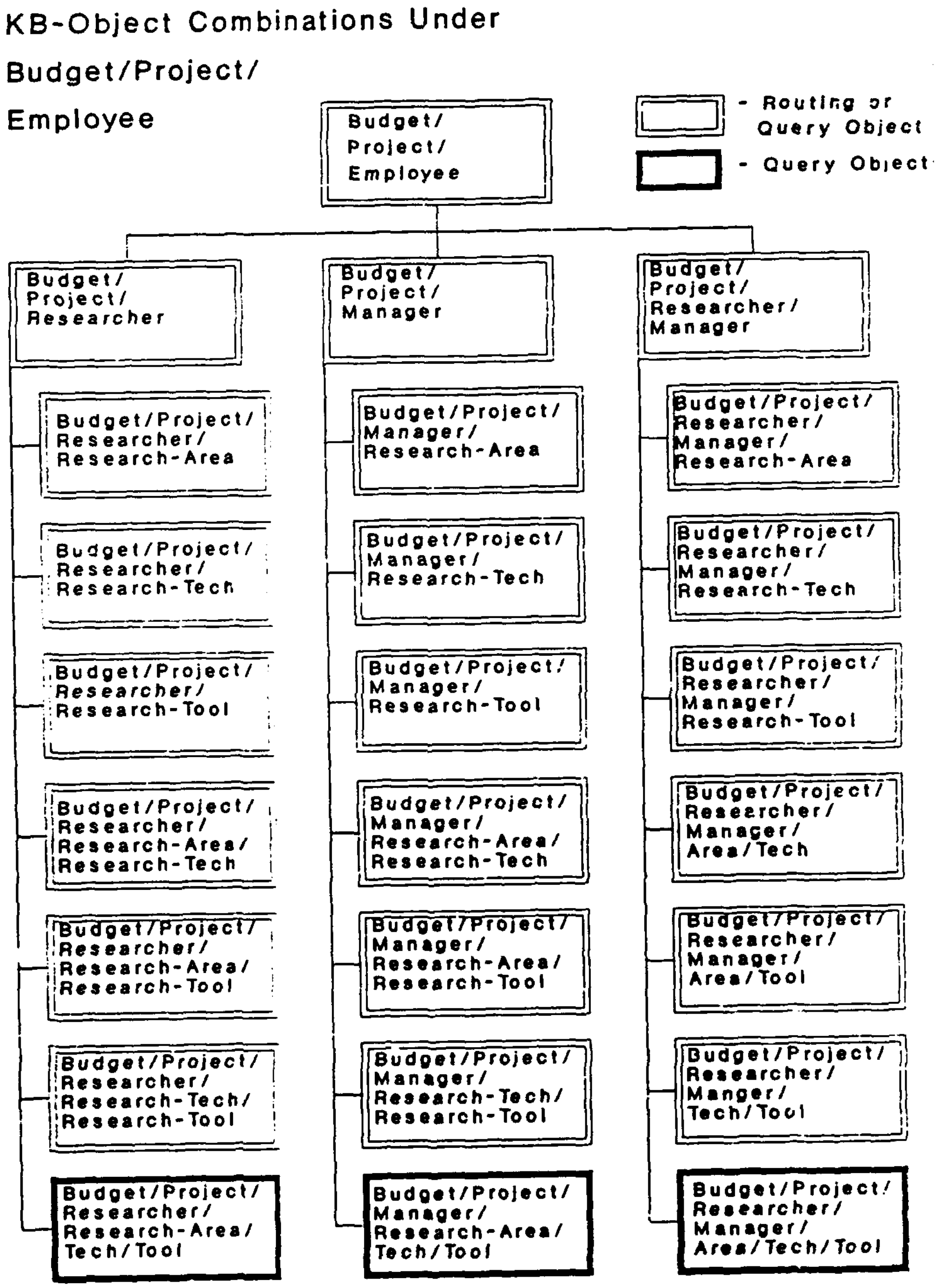

KB-Object Combinations Under Budget/Project/ Employee
Budget/ Project/ Employee
- Routing or Query Object
- Query Object
Budget/ Project/ Researcher
Budget/ Project/ Manager
Budget/ Project/ Researcher/ Manager
Budget/Project/ Researcher/ Research-Area
Budget/Project/ Researcher/ Research-Tech
Budget/Project/ Researcher/ Research-Tool
Budget/Project/ Researcher/ Research-Area/ Research-Tech
Budget/Project/ Researcher/ Research-Area/ Research-Tool
Budget/Project/ Researcher/ Research-Tech/ Research-Tool
Budget/Project/ Researcher/ Research-Area/ Tech/Tool
Budget/Project/ Manager/ Research-Area
Budget/Project/ Manager/ Research-Tech
Budget/Project/ Manager/ Research-Tool
Budget/Project/ Manager/ Research-Area/ Research-Tech
Budget/Project/ Manager/ Research-Area/ Research-Tool
Budget/Project/ Manager/ Research-Tech/ Research-Tool
Budget/Project/ Manager/ Research-Area/ Tech/Tool
Budget/Project/ Researcher/ Manager/ Research-Area
Budget/Project/ Researcher/ Manager/ Research-Tech
Budget/Project/ Researcher/ Manager/ Research-Tool
Budget/Project/ Reseercher/ Manager/ Area/Tech
Budget/Project/ Researcher/ Manager/ Area/Tool
Budget/Project/ Researcher/ Manger/ Tech/Tool
Budget/Project/ Researcher/ Manager/ Area/Tech/Tool

Figure 13

environment, stored in the *KB-Environment*, is restored.

Get-Display-Input retrieves the display file and field names chosen by the user, and assembles them into SQL syntax to be inserted later into the query. *Get-Search-Input* retrieves the search file and field names chosen by the user, and puts them in a SQL search statement consisting of the search file name, followed by all search statements associated with it, followed by the next search file name, etc.

4.4.3 KB-Environment The KB-Environment maintains a stack of state variables defining all environments created during a single query-building session. This stack is cleared at the beginning of each new query cycle, and an environment is saved immediately before the firing of a rule. If the user enters an invalid (unrelated) combination of keywords, an error message is given, and the previous environment is restored. Previous environments are also available to the user through the "BACK UP" command on the system menus.

4.4.4 KB-Macros A macro was written to define each KB-Objects instance; this macro calls a second macro that creates each instance's rule-list. The first macro is evoked by a single command containing the macro name followed by its arguments, entered either on the SCHEME command line or stored in a file that is then executed.

A portion of the code that defines the new object and creates one of its rules is as follows:

```
(macro make-leaf-obj (lambda(e)
  (let*(
      (obj-name          (list-ref e 1))
      (obj-d-list        (list-ref e 2))
      (obj-child-list    (list-ref e 3))
      (obj-file-name     (list-ref e 4))
      (df-obj            (list-ref e 5))
      (keys              (list-ref e 6))
      (link-file         (list-ref df-obj 2))
      (outfile           (open-output-file obj-file-name))
      (clause            '())
      (rule-name         '())
      (rule-count        1)
      )

  ;;; define object
  (set! clause
    '(define ,(eval obj-name) (make-instance kb-objects
      'name          ,obj-name
  'd-list         ,obj-d-list
  'child-list     ,obj-child-list
  'df-obj         ,df-obj
  'keys           ,keys
  'link-file      ,link-file
)))
(write clause outfile)

;;; define rule
(set! clause
  `(make-obj-rule
    'name ,rule-name
    'condition
      '(and   (eq-mem? user-list ',(eval obj-d-list))
                  (equal? run-query '())
                  (equal? more-cri '())
                  (equal? query-spec `()))
    'action
      '(begin
        (set! header "Specify An Action:")
        (if(not(equal?(send,(eval obj-name)get-child-list)'(EMPTY)))
          (set! menu-choices
            '(select-search/display-fields select-more-criteria))
          ;;; else
          (set! menu-choices '(select-search/display-fields))
        );end if
        (write-to-file header menu-choices)
        (dos-call "read.exe" (string-append "menu.txt "
          (convert-to-string user-list)))
        (send kb-translator get-input))))
(write clause outfile)
```

A sample macro call is:

```
(make-leaf-obj 'EMPLOYEE '(EMPLOYEE) '(RESEARCHER MANAGER STAFF) "EMP.S"
'(EMPLOYEE) '(SSN))
```

This command creates an instance of KB-Objects named EMPLOYEE, with the following instance variables:

Determinant-List	(EMPLOYEE)
Child-List	(RESEARCHER MANAGER STAFF)
Database file	EMPLOYEE
Key	SSN

This object is stored in the file "EMP.S" to avoid having to re-generate it each time the system is started.

5. FUTURE DIRECTIONS

A sample of additional features that can be added to PATHFINDER include:

- Queries joining four or more database files by using SQL views;
- Mathematically-derived query results;
- The ability to generate or validate project budgets by matching user-entered project parameters to previous projects, and comparing related budget items;
- Error analysis to determine why a query returned no values or unexpected values;
- Query syntax optimization to minimize retrieval time;
- A clear (non-SQL) statement of what a potential query will display, presented to the user prior to running the query;
- Allow the user to compose queries directly, in a more natural language than SQL, and then have the system translate the query into SQL syntax.

6. CONCLUSION

The PATHFINDER architecture is innovative because it is flexible, maintainable, and expandable. It demonstrates a way for novice users to choose from an exhaustive set of database queries without numerous lines of hard-coded and restrictive SQL statements. Due to the dynamic nature of the queries, the knowledge base is independent of the database content, and is easily modified if the database schema is changed. The knowledge base can also be expanded to a higher level of abstraction to incorporate different and more sophisticated types of knowledge.

Pathfinder also demonstrates that object oriented designs provide more than just an easier, more natural way to represent a program. Object orientation allows clear data models to be displayed to end users in an understandable way and improves their access to that data. The potential for improving database and knowledge base representation to both programmers and users by using the techniques described in this paper appear high.

REFERENCES

1. BRODIE, M.L., BALZER, R., WIEDERHOLD, G., BRACHMAN, R., MYLOPOULOS, J. Knowledge Base Management Systems: Discussions for the Working Group, Expert Database Systems, Proceedings From the First International Workshop, October 24-27, 1987, Kiawah Island, S.C., p. 24.

2. Bell, D.A., Shao, J., Hull, M.E.C., Integrated Deductive Database System Implementation: A Systematic Study, *The Computer Journal*, 33, 1, 1990.

3. WEIDERHOLD, G. Knowledge and Database Management, *IEEE Software*, Jan. 1984, 63-73.

4. GARDARIN, G., Valduriez, P. *Relational Databases and Knowledge Bases*, Addison-Wesley, Reading, Mass., 1989.

5. Al-Zobaidie, A., Grimson, J.B., Expert systems and database systems: how can they serve each other?, *Expert Systems*, 4, 1 (Feb 1987).

6. Potter, W.D., KDL-Advisor: a knowledge/data based system written in KDL, *in Proc. 21st Annual Hawaii Int. Conf. on System Sciences and Knowledge-Based Systems Track*, (Kailua-Kona, HI, Jan 1988).

7. Sheu, P.C., Describing Semantic Data Bases with Logic, *The Journal of Systems and Software*, (Sept 1989).

8. Smith, P.J., Shute, J., Galdes, D., Knowledge-Based Search Tactics for an Intelligent Intermediary System, *ACM Trans on Information Systems*, 7, 3, (July 1989) pp 246-270.

9. Smith, P.J., Krawczak, D.A., Shute, S.J., Chignell, M., Cognitive engineering issues in the design of a knowledge-based information retrieval system, *Proceedings of the Human Factors Society - 29th Annual Meeting - 1985* (Baltimore, Md., Sept 29 - Oct 3), pp 362-366.

10. Monarch, I., Carbonell, J., COALSORT: A knowledge - based interface, *IEEE Expert*, (Spring 1987).

11. HIGA, K., SHENG, O.R.L. An Object-Oriented Methodology for End-User Logical Database Design: The Structured Entity Approach *in Proceedings of Compsac '89* (Orlando, Fla., Sept. 1989).

12. CHEN, P. P. The entity-relationship model-Toward a unified view of data. *ACM Trans. Database Syst.*, 1, 1 (1976), 9-36.

13. ELIASON, A. L. *Systems Development - Analysis, Design and Implementation.* Scott, Foresman and Company, Glenview, Ill., 1987.

14. HIGA, K. End-User Logical Database Design: The Structured Entity Model Approach. *Unpublished Ph.D Dissertation, University of Arizona, 1988.*

15. Zeigler, B. P. University of Arizona, Electrical and Computer Engineering Department, Personal Communication.

IOIS SOFTWARE REQUIREMENTS

The implementation of an integrated knowledge base and database for the IOIS has introduced new software requirements. Previously, the only commercial package required to run the IOIS was EXSYS, an expert system shell. Most of the IOIS was coded in "C" and did not require runtime licenses or third party software. A detailed review of IOIS software and costs can be found in "Integrated Office Information System (IOIS) Interim Report" submitted to AIRMICS September 11, 1989.

With the addition of an "intelligent" database, two new packages are now required. Oracle was chosen for the DBMS after discussions with AIRMICS indicated AIRMICS had access to Oracle. EXSYS was not used for the knowledge base because it is not object oriented. Instead a dialect of LISP, PC-Scheme with SCOOPS, was chosen. PC-Scheme provides the power and flexibility of LISP with the addition of the Scheme Object Oriented Programming System.

Scheme provides a good research programming environment, however, it is an interpreted language and must be loaded before running a Scheme application. It is inexpensive, and can generally be found at mail order prices under $100 (the student version costs less than $40 and is all that is required to run the prototype - the only restriction is that it cannot be run in extended or expanded memory). PC-Scheme is not a particularly good language to use if the prototype is to be expanded into a full-sized system, however, since programs developed with interpreted languages tend to run slowly (the PC-Scheme knowledge base still runs faster than applications developed with Nexpert - see earlier IOIS reports). It would be easy to translate the PC-Scheme knowledge base using a language such as C++ into a compiled program. Less memory would be required to run the program (the PC-Scheme programming environment would not have to be loaded first) and the resulting program would run faster. A compiled C++ program could also be distributed without the complications of buying extra third party software or runtime licenses.

If the knowledge base must run in a UNIX environment, it would have to be recoded since the UNIX version of Scheme does not offer the SCOOPS enhancement. C++ offers an attractive option for doing this and would maximize portability.